<u>Conection between SAT3 and P throw match-N</u>

Juan Manuel Dato Ruiz
jumadaru@gmail.com

Abstract

In this study we will be able of finding a connection between SAT and P. This demonstration is constructive, so every concept will be used to generate a structure for solving any kind of problem. In half of the process of demonstration we will redefine the SAT formula like a problem of matching of some dimensions (as much as the number of clausules in the formula). So, we cannot ensure the connection of every formula well formed with P, and definitions exposed here will show the reason of why the last code halts always with a solution.

<u>Part 1. General terminology</u>

The reason of this part is that in every theory of computer science the same definitions will always appear at the very begining for solving each class of problem. This part will be rewritten in the future with new terminologies which will connect energy, entropy and data, but essentially every definition shown here will get no changes.

Def 1.1. A <u>data</u> is a symbol (a distinguishible value) or a token of our alfabet whose value is ordinal: if we have two datas, then we will count with *data first* and *data second*. That will be exactly the same of saying we have a *data second* and *data first*; as saying as we have two datas popped from a container with more datas.

Note: A data does not have to be a number, because numbers are necesary ordered (0<1<2<...), but data not.

Def 1.2. A <u>name</u> is the information we give to a data: the sense or reason of his recognizement. When we give a name to a data, data will get a meaning for all of us.

Note: A name is not a data, because we do not use it like that; however it could be possible to define a <u>metadata</u> like the data which stores a name. That is like giving names to the variables of an algorithm. I can use the symbol 1 to represent a “tomato”, tomato is the name and 1 is the data.

Def 1.3. A <u>tuple</u> is a data which contains ordered datas. Each of these datas contained in the tuple will be called <u>components</u> (componentes). The number of components in the tuple will be called <u>length of the tuple</u>. The component number **i** will correspond to the only one data after **i** datas into the tuple.

Def 1.4. A matrix is a data which contains ordered tuples of the same length. So, that is an ordered set of tuples of the same length. Each tuple is called row, and the tuples formed by data in the same component will be called columns.

Def 1.4.1 The transpose of the matrix is the result of redefining the matrix transposing rows with columns.

Def 1.5. A pattern is a data which show us if a tuple is in a set. From a pattern the format is interesting: that is the way we will understand what tuples are in and what out.

Def 1.5.1 A pattern could be inclusivist when only shows the tuples which are in the set.

Def 1.5.2 A pattern could be exclusivist when only shows the tuples which are out the set.

Note: To define the format, we have to use some spetial symbols which represents concret things, or else enumerating the tuples one by one explicitly.

Part 2. More specific terminology.

From the play of the problem of correspondences we will have to define a more expert terminology. It will help us to expert areas of study.

Def 2.1 The permutation of a tuple is the copy of the tuple with each of their datas reordered.

Def 2.2 A tuple compounded by all distinct data can represent the information of a permutation.

Note. Here we would say a tuple can be used like metadata of how to permutate tuples. Is easy imagining ways, and the objetive of the expert is to choice a way.

Def 2.3 A model is a way of interpretation without contradictions of reality. That is so, a kind of information.

Note. We have to understand the model like "a model of reality", or like an acceptable instance. Where the instance of reality has the meaning of being the only one in the configuration of reality.

Def 2.4 A relation between two datas is a tuple formed by two datas into a model. We can write the relation between a and b like **a R b**, or more shortly **ab**.

Def 2.4.1 When a relation **a R b** is <u>of equivalence</u> we will say we can define a model where both datas get the same meaning. Then we will can write it in this way: **&a = &b**.

Note: The model we will use can be full of names, and one of those names can represent the meaning of both datas. For that reason, it can be understanded like the "direction" (the metadata of the name) of both datas in this model is the same.

Prop 2.4.2 If **ab** is relation of equivalence then **ba** is relation of equivalence.

Dem 2.4.2
From def 2.4.1, we can define a model of reality where a and b mean the same thing; so we can define a model of reality where b and a mean the same thing.
□

Prop 2.4.3 if **&a=&b** and **&b=&c** then **&a=&c**

Dem 2.4.3
&a=&b means there is a metadata with the meaning of **a** and **b** in the model. This metadata is used for **a** so much so for **b**.
If that metadata is different to the same used for **a** and **c**, then the model will get two different names for the same meaning in **b**. So reading def 2.3, there cannot be two different meanings to **b**.
□

Note: We have to understand the relation like a view of the datas, not like an assignment. The sense and the meaning we give something if it is duplied then its store in metadata will be an overcost and the contradiction of saying there are two meanings different to the same stored meaning.

Def 2.4.4 We can note: No **&a=&b** equals to **&a!=&b**. In a model cannot be possible a relation and its opposite.

Prop 2.4.5 &a=&b and **&a!=&c** implies **&b!=&c**

Dem 2.4.5
2.4.5.1. &a=&b
2.4.5.2. &a!=&c
2.4.5.3. Sup &b=&c
2.4.5.3.1 &a=&c by 2.4.3, 2.4.5.1, 2.4.5.3
2.4.5.3.2 Contradiction by 2.4.5.3.1, 2.4.5.2
2.4.5.4 &b!=&c
□

Def 2.4.6 Instance of the name is the tuple which represents the multiple relation of equivalence between its components.

In example: If **abc** is an instance of the name then we will can write it in this way too: **&a=&b** , **&b=&c**, **&a=&c**

Def 2.4.7 We will call relational pattern if his format is defined by relations. The accepted tuples by the pattern will contain their instances of the name in their respective components.

In example:

?	A	?	?	B	c	?	?	D	?
a	A	b	b	B	c	a	c	D	A

The tuple is accepted in the instance of the name of the relational pattern, because each component can be accepted in the pattern.

Def 2.4.8 We will define the projection pattern on a tuple like the application of the pattern on the components of the tuple where the tuple is defined.

Note: In this way, if the pattern obligates that a component gets a value, then if the tuple does not aply on that component then the requirement will be ignored.

Part 3. Definition of the problem

Now we are going to define the problem of match-N. Thanks to his definition it will be easy for us seeing how works the polynomial resolution in a Turing Machine Deterministic for solving some Non Deterministic Turing Maching of Polynomial Resolution problems.

Def 3.1 A match-N is a matrix of q tuples of length N defined into an alfabet of q datas where those datas are defined in a relational pattern P and his traspose is a matrix of permutations.

Def 3.2 A submatch-2 is a submatrix of q tuples defined into a match-N by two different components and we can write it like **SM(i,j)** where $i<j$.

Note: The question $i<j$ is not for avoiding **SM(j,i)**, the fact is that we will use it like **SM(j,i)** = **SM(i,j)**

Def 3.2.1 Keeping the idea, we can grow the notation to use $SM(i_1,i_2,...,i_n)$ like a submatch-n submatrix of q tuples.

Prop 3.3 In a **match-N** we can define **N*(N-1)/2** different **submatch-2** which are deffinible with a projection of the pattern of **match-N** on each **submatch-2** for generating independent solutions.

Dem 3.3.
3.3.1 A=match-N=matrix of q tuples of length N.
3.3.2 A=matrix q rows N columns , by 3.3.1
3.3.3 define $A_{i,j}$=SM(i, j) of A , by 3.2
3.3.4 It will happen:
1. The number of different $A_{i,j}$ is N*(N-1)/2, because they (for each pair i and j) constitute a structure of triangular number of order N.
2. Each transport of $A_{i,j}$ are two permutations, because columns are not modified.
3. The pattern of $A_{i,j}$ will be the same with the pattern applied on A; so the model will be the same after applying independient solutions.
□

Prop 3.4 The resolution of **SM(i,j)** of **A** and of **SM(j,k)** of **A** implies the resolution of **SM(i,j,k)**

Dem 3.4
SM(i,j) sets a relation of equivalence between **i** and **j**, as **SM(j,k)** as. Defining instances of name compatibles with only one pattern definied in **SM(i,j,k)**.
So,
3.4.1. &i=&j because **SM(i,j)** is a success
3.4.2. &j=&k because **SM(j,k)** is a sucess
3.4.3. &i=&j=&k by 3.4.1, 3.4.2
□
Note: If the instances of name definied by all the submatch generate two distinct models then they will cannot be the result of the projection of the same pattern. In that case, **j** must be associated with two instances of name for the same model of the reality, that is impossible.

Prop 3.5 The construction **match-N** can be got in time and space polynomial respecting to **N**.

Dem 3.5
By 3.3, we need a quadratic quantity of calculations respecting to **N**, and by 3.4, that will be enough.
□

Prop 3.6 The construction of **match-N** will be solved in time and space polynomial respecting to the pattern.

Dem 3.6
The way we have to define the pattern never will need more than the number of datas contained in all the matrixs. The number of datas is so in an order of $(\mathbf{qN})^2$, for a **match-N** of **q** datas.

Considering the construction of the solution is quadratic itself (by 3.5), then defining the pattern will need more costs than finding the solution.
□

Part 4. Equivalences with satisfiability
I will resume and analyse every step for that be the most clear possible.

Def 4.1 A formula in **SAT3** is composed by a set of clausules which are composed by 3 literals, which are variables we can negate or not.

Def 4.2 They say an assignment of values on the variables satisfies the function of **SAT3** when the result of applying the values of truth on literals added in each clausule will be always **True** in every one.

Prop 4.3 For a resolution, each clausule will be labeled by a number 1, 2 or 3; this one will set that literal is the literal that will make the clausule **True**, when the rest of literals will be ignored.
Dem 4.3
A possible assignment of each literal implies the literal will be true so, in the clausule, (if it could be done for each "main" literals for each clausule) there will be an assignment of variables. However, if it is not possible we won't find a tuple with 3 kind of symbols.
□

Prop 4.4 The assignment to a literal negated or not of a variable to a value (**True** or **False**) implies the elimination of every clausule one by one with the same literal for solving the same problem.

Dem 4.4
If the clausule $\mathbf{C_j}$ is set by the literal negated $\mathbf{X_i}$ to be **True**, then if we cannot ignore the $\mathbf{C_k}$ which include $\mathbf{X_i}$ negated we will including exact clausules but with $\mathbf{X_i}$ not negated; so the problem will restrict solutions.
Considering the literal not negated, or the value **False** will construct an analagous demonstration.
□

Def 4.5 A subtable is a no-final structure which is needed for the simulation of all possible no-final queries before defining completely a submatch. So the subtable will be defined as a matrix of 3 rows and 3 columns.

Def 4.5.1 Each data will be stored in one of the 9 celds (with coordinates row and column in the table **i**, **j**) and it will be definied exclusively in four queries: activated, unactivable, accepted e rejected.

Def 4.5.1.1 <u>Activated</u> means the names related to their coordinates will be choiced (considering 4.3): giving a solution to the clausules with those coordinates.

Def 4.5.1.2 <u>Unactivable</u> means that celd will never be activated. That celd could be valued like accepted of rejected.

Def 4.5.1.3 <u>Accepted</u> means generates a correspondence between row and column. The active celd implies that is accepted too.

Def 4.5.1.4 <u>Rejected</u> means the opposite of accepted.

Def 4.5.2 A subtable can be in one of four queries: unactivated, activated, complete y nulled.

Def 4.5.2.1 <u>Unactivated</u> means in that subtable there is not a celd activated yet. Only from this state a celd can be activated, but not if subtable is nulled.

Def 4.5.2.2 <u>Activated</u> means there is a only one celd that is activated. Only from this state it can go to the state complete.

Def 4.5.2.3 <u>Complete</u> means for each row or column there is only one accepted celd and the rest rejected. This implies the subtable represents a permutation between names of rows and the names of the columns. So that means we have got a solution of the submatch.

Def 4.5.2.4 <u>Nulled</u> means the subtable is deleted because almost one of their clausules has a solution and if we keep the subtable activated it could restrict the number of solutions. Like it was set in the proposition 4.4.

Def 4.6 I will call <u>triangle table</u> (**TT**) to the set of the subtables representing all **submatch-2** which solve the problem of satisfiability. In this way, the subtable **i**,**j** in **TT**, will correspond with clausules $\mathbf{i}^{\mathbf{th}}$ and $\mathbf{j}^{\mathbf{th}}$.

Prop 4.6.1 When we activate a literal we have to null every subtables with that name.

Dem 4.6.1
It can be understood by proposition 4.4 and definition 4.5.2.4
□

Prop 4.6.2 When a literal is activated (from the activation of a subtable) we have to unactivate every celd which name is the same with the literal but negated.

Dem 4.6.2
Suppose a literal is activated and his negated,
Then we are assigning to a literal a True and a False values
Contradiction
So we cannot activate this one with its negated.
□

Prop 4.6.4 With the acceptation (or activation) of a celd in the subtable **i**, **j** in coordinates **f**, **c**, and there is a subtable **i**, **k** whose celd with coordinates **f**, **c** is rejected, then we will reject (logically) the celd of the subtable **j**, **k** in coordinates **f**, **c**; and if it is accepted in **i**, **k** we will accept it in subtable **j**, **k**.

Dem 4.6.4
The link between the subtables and the submatch is exactly the link between **TT** and the **match-N**. So it is explained in proposition 3.4.

In that way, if we get two subtables with a clausule in common which have accepted a celd in the same coordinates, then for each subtable not nulled if that coordinate is not accepted it will go to say one assertion (meaning) has two possible values.
□

Prop 4.6.5 Choicing a celd to activate between some will generate a logic supposition. If **TT** generates an inconsistent query then the supposition fails.

Dem 4.6.5
The process of activation the subtables is equivalent to activating the clausules. So, if any clausule cannot be activated that means that some queries finished in a forbidden step to get a state where the table means something inconsistent.

That is the reason it goes to the evaluation of insatisfiability of the set of the clausules.
□

Prop 4.6.6 If we begin an inconsistent supposition **TT** will finish finding the false step without choicing other celd for another supposition.

Dem 4.6.6

To get a subtable would be enough to generate a contradiction, it needs a rejected celd in a literal choiced. In fact, activating a literal, if that subtable depends of the launched supposition by the activation of the literal, then it will finish rejected. So, in the columns or rows where we can choice where to activate, there will be only one space (celd) where do it (or no one).

If we finish in a new choice that is because it is not a deppendable one, so up to this we cannot got a contradiction and, choicing what we want, it will never go to a contradiction. In this case, we could say without the set of assignments we got the resolution of the problem of satisfiability.

□

Def 4.6.7 The tray of the activated if a imaginary line that joins the coordinates activated jumping from subtable in subtable with a clausule in common for activating all the clausules without a repetition.

Example: If the coordinates **f**, **c** were activated in the subtable **i**, **j**, then the tray will begin in **f**(**i**), **c**(**j**), and it could continue changing or **f**(**i**) or **c**(**j**) but not both of them. Forming like a domino:
(**1**(**2**), **3**(**5**)) - (**2**(**6**), **3**(**5**)) - (**2**(**6**), **2**(**1**))

Prop 4.6.7.1 The tray of the activated generates an instance of name that represents the assignment which satisfies the clausules.

Dem 4.6.7.1
We know the tray of the activated represents the resolution of the pair of clausules it contains.
If by hypothesis, a tray of length **N** solves a **submatch-N** then the next one in the tray will be a distinct clausule that will be activated like in proposition 4.6.6 was shown. So the tray of length **N+1** will solve the **submatch-N+1**.

□

Def 4.7 I will show you an enough good algorithm for understanding why it is polynomial solvable.

4.7.1 Ordinary Steps
4.7.1.1 Mount the table **TT** where for each pair of clausules **i** and **j** the celds will be rejected if literals are under contradiction.
4.7.1.2 A subtable is choiced for activating. It will be called **A**.
4.7.1.3 A celd is choiced for activating. It will be called **C1**.
4.7.1.4 The subtable is deffended of inconsistences.
4.7.1.5 Subtables are nulled by 4.6.1
4.7.1.6 Celds are unactivables by 4.6.2
4.7.1.7 The **TT** is deffended of inconsistences by 4.6.4
4.7.1.8 The tray is grown to get the next one for activating by 4.6.7.
4.7.1.9 Jump to 4.7.1.3
4.7.2 Alternative steps
4.7.2.1 If fails **TT** in 4.7.1.4 or 4.7.1.7 then another celd of **C1** in the same row is choiced. It will be called **C2**. Jump 4.7.1.4
4.7.2.2 If fails steps in 4.7.2.1, 4.7.1.4 or 4.7.1.7 then the other celd of **C1** and **C2** in the row is choiced. It will be called **C3**. Jump 4.7.1.4
4.7.2.3 If fail steps in 4.7.2.2, 4.7.1.4 or 4.7.1.7 then we will declare the insatisfiability of the formula.
4.7.2.4 If fails in 4.7.1.8 then all clausules will be activated: It will be declared the satisifiability of the formula. And the marked celds show the assignments for its resolution.

Prop 4.7.3 The algorithm definied in 4.7 has polynomial resolution in space and time.

Dem 4.7.3
For an entry of **C** clausules this algorithm needs up to **C*(C-1)/2** subtables to fill the triangle table. Each subtable needs 9 celds. So it is needed exactly **9*C*(C-1)/2** celds of works.

In other hand, we only launch 3 suppositions (**C1**, **C2**, **C3**) to fill this space (by proposition 4.6.6). So the number of steps is proportional to $\mathbf{C^2}$.

□